Control of an Inverted Pendulum

Christian Chang, Aaron Erickson, Joshua J. Gross, and Will Yingling
Robotics Engineering Department, Worcester Polytechnic Institute
Worcester, Massachusetts, USA
*cmchang@wpi.edu †aberickson@wpi.edu, †jjgross@wpi.edu, ‡wpyingling@wpi.edu

***Abstract*—This paper presents a study on implementing feedback controllers for a single link inverted pendulum system, evaluating seven unique controllers for their performance. The system was modeled using Simulink and Simscape. Factors such as rise time, settling time, and overshoot are used to evaluate error based controllers as well as more advanced experimental controllers. The process of modeling the system as well as each controller is explained in detail along with the respective results. Conclusions are drawn as to which controller performed the best as well as ways to improve with more research.**

***Keywords*—Inverted Pendulum, Robotic Control, Dynamic System**

I. Introduction

The following offers an insight into the design and control of an inverted pendulum. The inverted pendulum, being a second order dynamical system offers the opportunity to implement controllers into the DC motor, a cart and track system, and a link itself. An example of an inverted pendulum can be seen below in Fig. 1 [1].

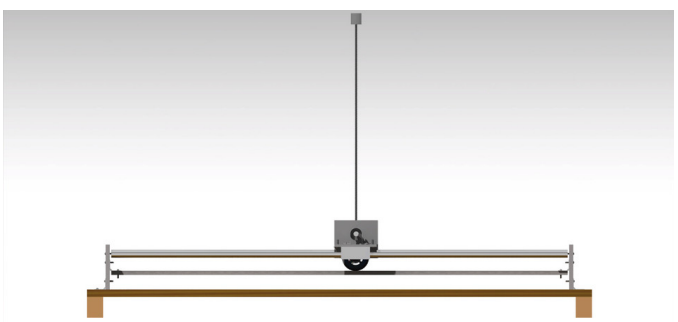

Fig. 1. Inverted Pendulum on Cart CAD Design

The dynamics of the system are evaluated and then used in the evaluation of the system by means of implemented controllers. The creation of the system is based in programmatic tools such as MATLAB, Simulink, and Simscape Multibody. The following sections explain the derivation of each element of the system as well as their respective creation in Simulink utilizing Simscape multibody.

The following controllers are implemented into the schematic in order to modify the system's behavior:

1) Proportional (P)
2) Proportional and Derivative (PD)
3) Proportional and Integral (PI)
4) Proportional, Integral, Derivative (PID)
5) PD plus Feedforward
6) Linear-Quadratic Regulation (LQR)
7) Model Predictive Control (MPC)

Each controller is also explained in detail regarding the mathematical derivation and their implementation into the system. Note that the LQR and MPC controllers are outside of the scope of this effort and therefore are primarily supported by theory with some implementation.

Once a system capable of consistent testing is constructed, the different controls are applied and their individual performance results are compared to each other to determine which is the best in this specific application.

II. Background and Supporting Literature

In order to approach this system, research was conducted into previous methodology for modeling the system as well as implementing controllers. Although previous knowledge of error based controllers such as a PID controller [2] is utilized, research suggests that using a more complex controller will offer optimal results. Due to the inverted pendulum being a classic controls problem, there is an abundance of research and suggestions as to the best methods for approaching this second order under-actuated system.

Multiple sources suggest exploring a H2 controller for full state feedback [3]. This controller presents a gain value that attempts to balance the plant while minimizing the cost function. While interesting, it was determined that this controller was not ideal for this system as the H2 controller is designed for linear system and the construction appeared to be too similar to an error controller.

Other sources suggested the use of model predictive control in order to balance the inverted pendulum system. MPC is an "optimal control technique in which the calculated control actions minimize a cost function for a constrained dynamical system over a finite, receding, horizon" [6]. This controller works by using the current state of the system to calculate and predict future states. This information is then used to make a decision as to the best next action to balance the system. Due to the interesting nature of this controller, MPC was selected to be implemented in an exploratory role in this effort.

Another common suggestion for controlling an inverted pendulum system is to utilize a linear-quadratic regular (LQR). A LQR controller "can provide a local approximation of the optimal control solution for the nonlinear system" [4]. The LQR controller utilizes multiple mathematical expressions to linearize a nonlinear system in an attempt to balance and reach

steady-state. This controller is very common in under-actuated, non-linear systems such as the inverted pendulum. Due to this reason, the LQR controller was also selected to be further explored.

III. Materials and Methods

The following discusses and explains the process that was followed in order to accurately create and evaluate the second order dynamical system as well as each controller that was later implemented. In practice, the following tools were used in order to achieve our desired results: MATLAB, Simulink, Simscape Multibody, Simscape Electrical, and Solidworks.

A. Dynamical Analysis

The first step in the process of modeling the second order pendulum system is to evaluate the dynamics of the system. This particular system consists of a single link attached to a cart that is capable of moving side to side when an external force is applied. Using Fig. 2 [2], the total energy of the system can be calculated in order to then solve the Lagrangian.

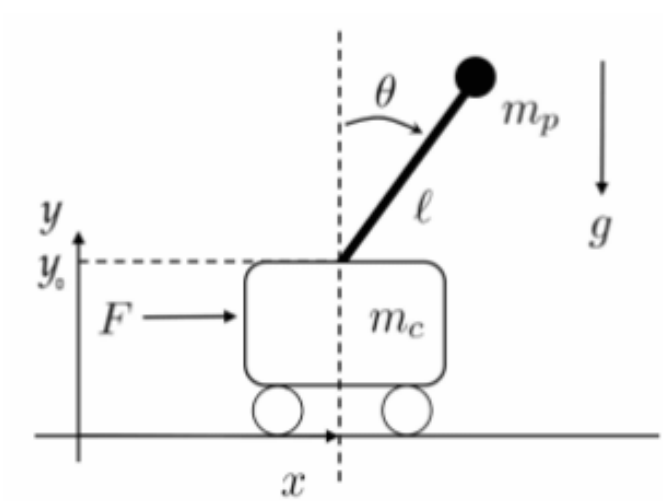

Fig. 2. One Link Inverted Pendulum Diagram

Combing the kinetic energy of the car moving sideways with the velocity of the link in motion yield Eqn. 1 [5]. The kinetic energy of the link's movement must be broken down into the horizontal and vertical component as seen.

$$T = \frac{1}{2}Mv_1^2 + \frac{1}{2}mv_2^2$$

$$T = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m((\dot{x} - l\dot{\theta}cos(\theta)^2) + (-l\dot{\theta}sin(\theta)^2)$$

$$T = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2]\dot{\theta}^2 - ml\dot{x}\dot{\theta}cos(\theta) \tag{1}$$

Assuming that the cart is resting on a stable surface of height $y = 0$, the potential energy of the system is limited to the mass of the link acting with gravitational force. This relationship is seen below in Eqn. 2 [5].

$$U = mglcos(\theta) \tag{2}$$

Now combining the potential and kinetic energy terms, the Lagrangian can be formed by taking the difference of kinetic and potential energy. This mathematical connection is expressed in Eqn. 3 [2]. Once the Lagrangian is formed, partial derivatives with respect to the linear and angular components, such as those seen in Eqn. 4 [5], of the system will yield the equations of motion that this inverted pendulum have.

$$L = KE - PE = T - U$$

$$L = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 - ml\dot{x}\dot{\theta}cos(\theta) - mglcos(\theta) \tag{3}$$

$$\begin{aligned} \frac{\partial L}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}} &= F \\ \frac{\partial L}{\partial \theta} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} &= F \end{aligned} \tag{4}$$

Once the partial derivatives have been taken and each equation has been simplified, Eqn. 5 and Eqn. 6 [5] are formed therefore expressing the equations of motion for this system. Both the force being applied to the cart as well as the acceleration of the inverted link can be expressed using these two equations.

$$F = (M+m)\ddot{x} - ml\ddot{\theta} \tag{5}$$

$$l\ddot{\theta} = g\theta + \ddot{x} \tag{6}$$

These equations of motion will be used later in order to accurately model the system as well as create the controllers appropriately for tuning to the point of stability.

B. Experimental Setup

To approach modeling this system programmatically, the structure of the problem must first be understood and designed. Fig. 3 below shows the generic structure used for creating each element of the system and seeing how all the different components work together.

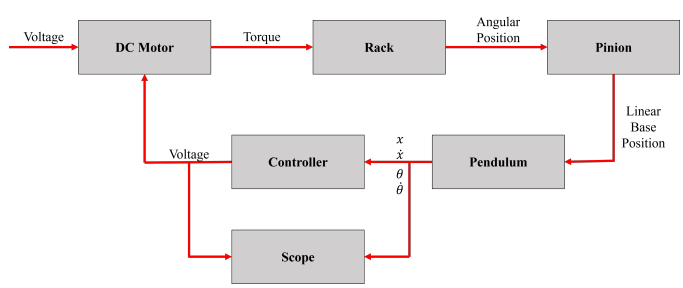

Fig. 3. Experimental Setup

As seen in the above figure, the system input is voltage which is used to power a DC motor. The DC motor outputs torque and applies it to the track system, which is decided to

be a rack and pinion type of system for translating angular to linear motion. The rack and pinion blocks will combine to output a position of the base of the inverted pendulum. This motion is translated into linear and angular motion by the link where the respective positions and velocities are outputs. These values are plotted but are additionally fed into the arbitrary controller. The controller will then output a new voltage value that is fed back into the DC motor, therefore closing the loop and creating a closed-loop feedback system.

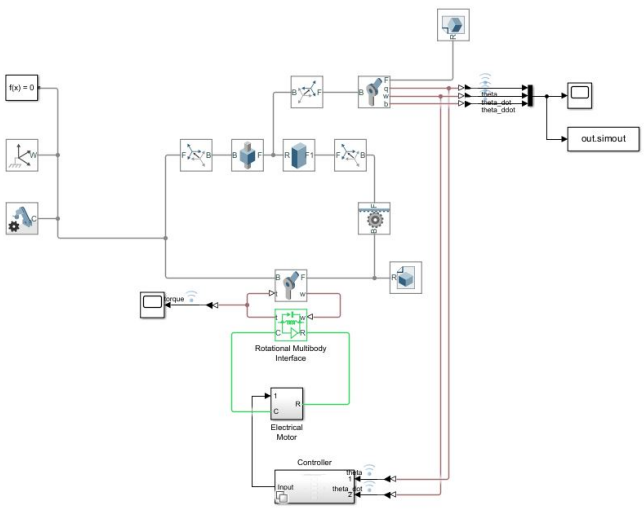

Fig. 4. Experimental Setup Modeled in MATLAB

This Simulink model includes file from Solidworks in which the material properties are contained. Additionally, using Solidworks, a track system for our system to operate on was constructed allowing the performance to easier be observed. Fig. 5 shows this constructed system, a visual representation of the Simulink model. As seen in the CAD drawing, a track system was created for the base cart to move along horizontally while the pendulum link is attached to the cart by means of a free spinning pin. The circle on the far right of the figure represents the system in which the cart will be moved (rack and pinion or gear and pulley).

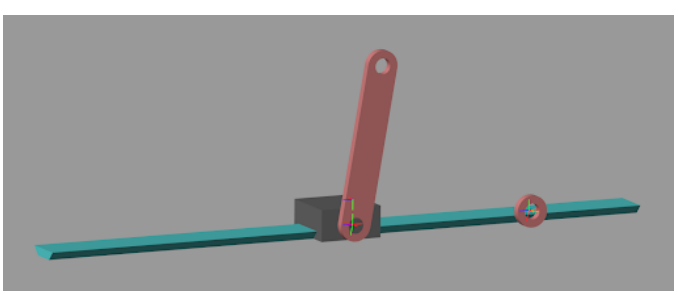

Fig. 5. Solidworks Model of Entire System

Now that the general structure is understood, each component is modeled in Simulink utilizing Simscape multibody. Fig. 4 above shows the overall system which was created. The following sections explore each individual component of this model. While the model is truly much more intricate than the one shown above, subsystems were used and implemented for sake of clarity and organization.

C. DC Motor

The DC motor was modeled using the Simscape Electical in addition to a Simulink subsystem. This subsystem accepts voltage as an input which is in turn run through a closed circuit consisting of a resistor and ground. This voltage is then fed into a generic electric motor which is connected to a spring and damper system for stability purposes. The output of this motor is then fed into a rotational multibody interface which is capable of accepting the rotational element of the motor and applying it to a mechanical revolute joint. Fig. 6 below shows this subsystem fully constructed.

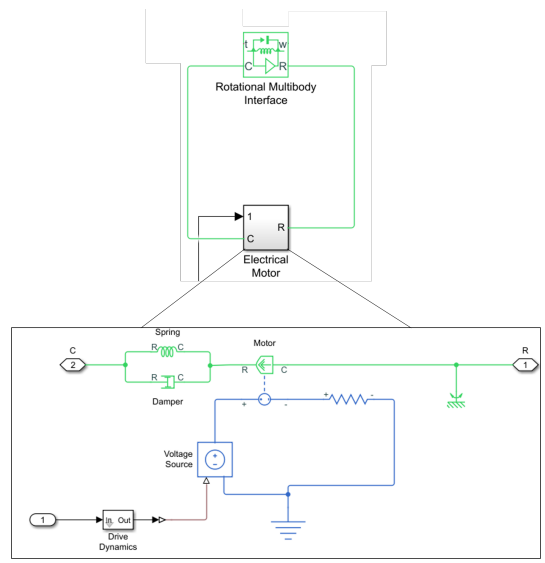

Fig. 6. DC Motor - Simulink Representation

D. Track System

As previously mentioned, it was decided to use a rack and pinion system for moving the cart horizontally to change the location of the pendulum's base. This system accepts the rotational output of the motor and applies it to a revolute joint where the rack is connected. A prismatic joint (the pinion) then in turn accepts the rotation of the rack and converts it to linear motion. A solid file was created to act as the connective piece between the revolute joint and the rack, making the system mechanically sound. Fig. 7 below shows the construction of the track system using Simscape Multibody elements.

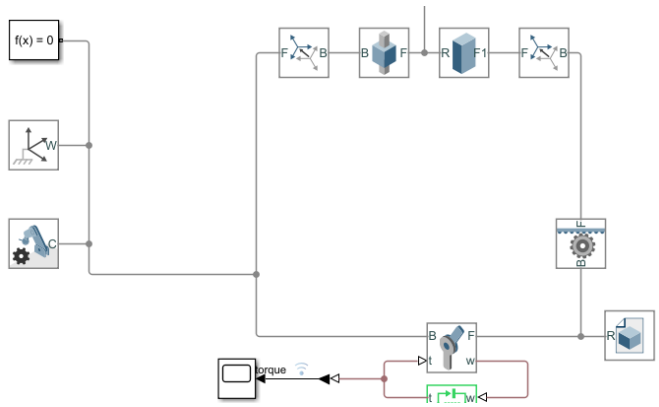

Fig. 7. Base Cart on Track - Simulink Representation

E. Inverted Pendulum

In order to create the actual link that acts as the pendulum, a solid was created in Solidworks where the properties of the link are set to be aluminium. Simscape Multibody is capable of accepting this part and calculating the mass and moment of inertia automatically. The position of the base is translated to this part where the linear and angular positions and velocities are the outputs which are sent to a scope for plotting. Fig. 8 shows this system as modeled with Simscape Multibody.

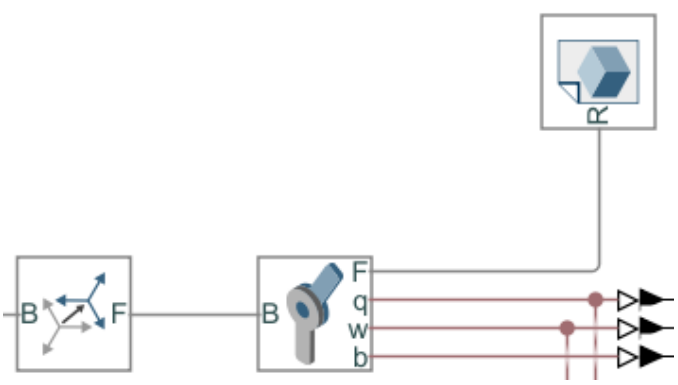

Fig. 8. Pendulum Link - Simulink Representation

F. MATLAB Inverted Pendulum

As elaborated upon later, a pre-modeled system of an inverted pendulum from MATLAB was used in order to accurately assess the performance of an MPC controller.Similar to the diagram previously shown, Fig. 9 [6] shows the system that was used in the creation of the pendulum system for the MATLAB example.

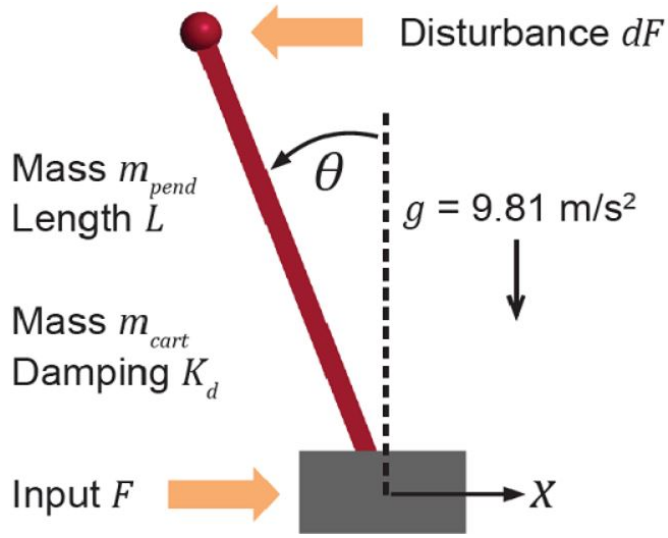

Fig. 9. MATLAB Inverted Pendulum Diagram

While significantly more complex, Fig. 10 [6] shows the modeling of the same inverted pendulum system. It should be noted that MATLAB solely utilized Simulink blocks and did not include any Simscape Multibody or Simscape Electric elements therefore further complicating the model. This system accepts a force input and outputs the same linear and angular position and velocity values. This entire model acts as the plant of the feedback loop which is later explored.

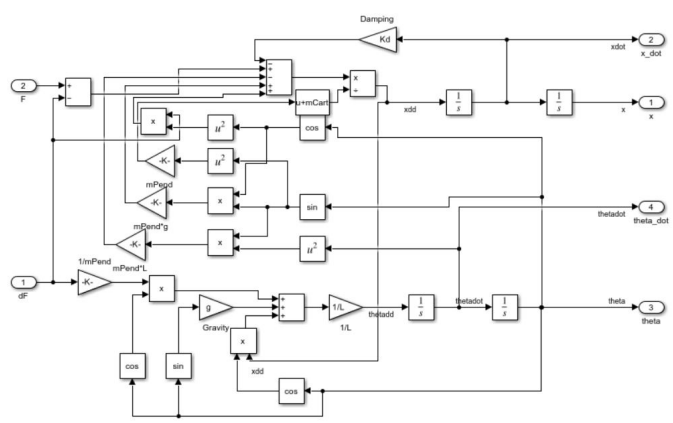

Fig. 10. MATLAB Inverted Pendulum Simulink Model

G. MATLAB Plant with MPC Controller

Now that the plant, or the inverted pendulum itself, has been created, it can be applied to a feedback loop where a controlled can be applied. In this case, an MPC controller is placed into the system. Since MPC controllers are outside of the scope of this effort, MATLAB's example is once again used for comparison purposes. This MPC block is placed into the feedback loop where the measured outputs (linear and angular position) and a reference value are fed into the model

predictive control block. Fig. 11 [6] shows the entire feedback loop including both the plant and the MPC block.

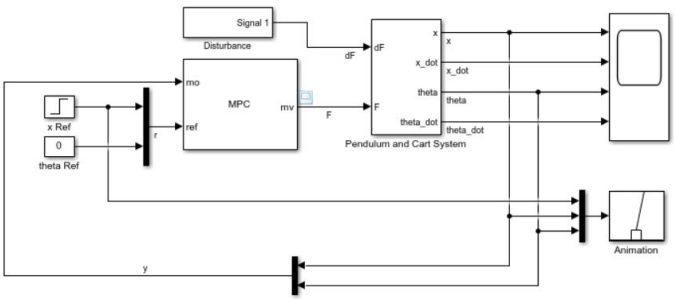

Fig. 11. MATLAB Inverted Pendulum with MPC Simulink Model

H. Controllers

The following sections explore the theory and construction of each individual controller so that they may be implemented into the system. Once all of the controllers are designed, they are implemented individually through subsystems. In order to make the simulation run smoothly, these subsystem controllers are added to a variant subsystem where the associated MATLAB script runs the simulation with each controller being applied. The following code snippet shows how the script will parse through the controllers and run the simulation using each.

```
VSS_MODE = 1;
InvertedPendulumWithMotor([], [], [], '
    compile')
InvertedPendulumWithMotor([], [], [], '
    term')
set_param('InvertedPendulumWithMotor', '
    StopTime', '5'); % can't have .slx
out = sim("InvertedPendulumWithMotor.slx
    ");
fprintf("Running P controller subsystem
    1/5 \n");
pause(6)

VSS_MODE = 2;
InvertedPendulumWithMotor([], [], [], '
    compile')
InvertedPendulumWithMotor([], [], [], '
    term')
set_param('InvertedPendulumWithMotor', '
    StopTime', '5'); % can't have .slx
out = sim("InvertedPendulumWithMotor.slx
    ");
fprintf("Running PD controller subsystem
    2/5 \n");
pause(6)

VSS_MODE = 3;
InvertedPendulumWithMotor([], [], [], '
    compile')
InvertedPendulumWithMotor([], [], [], '
    term')
set_param('InvertedPendulumWithMotor', '
    StopTime', '5'); % can't have .slx
out = sim("InvertedPendulumWithMotor.slx
    ");
fprintf("Running PI controller subsystem
    3/5 \n");
pause(6)

VSS_MODE = 4;
InvertedPendulumWithMotor([], [], [], '
    compile')
InvertedPendulumWithMotor([], [], [], '
    term')
set_param('InvertedPendulumWithMotor', '
    StopTime', '5'); % can't have .slx
%set_param('InvertedPendulumWithMotor/
    Controller/PID', 'VariantControl', '
    VSS_MODE == 4')
out = sim("InvertedPendulumWithMotor.slx
    ");
fprintf("Running PID controller subsystem
     4/5 \n");
pause(6)

VSS_MODE = 5;
InvertedPendulumWithMotor([], [], [], '
    compile')
InvertedPendulumWithMotor([], [], [], '
    term')
set_param('InvertedPendulumWithMotor', '
    StopTime', '5'); % can't have .slx
out = sim("InvertedPendulumWithMotor.slx
    ");
fprintf("Running FFW PID controller
    subsystem 5/5 \n");
pause(6)
```

1) Proportional (P)

The proportional gain, an algebraic gain implementation, applies a gain to the position signal and adjusts the voltage accordingly. This relationship is expressed by the simple equation below, Eqn. 7 [2].

$$D_{CL}(s) = K_p$$

$$u(t) = K_p * e(t) \tag{7}$$

Fig. 12 shows this controller being created in Simulink where it is then implemented into a subsystem and applied to the feedback loop in the controller position.

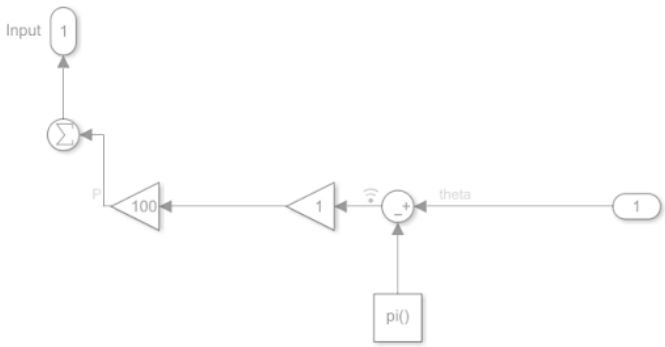

Fig. 12. Proportional Controller

2) Proportional and Integral (PI)

Similar to the proportional controller, the PI controller adds a term to account for the integral term of motion, or more commonly known as the acceleration. In the time domain, this is compensated for by applying a $\frac{1}{s}$ term. However, in the time domain, the error as a function of the force must be integrated to account for this term as shown in Eqn. 8 [2].

$$D_{CL}(s) = K_p + K_i \frac{1}{s}$$

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, dt \tag{8}$$

Fig. 13 shows this PI controller modeled in Simulink where it is later applied to the feedback loop. Note that the input signal is offset by π radians to account for the difference in angle of the link not following standard convention.

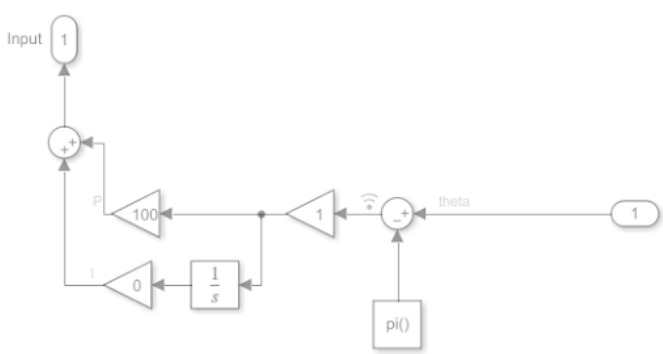

Fig. 13. Proportional and Integral Controller

3) Proportional and Derivative (PD)

The proportional derivative gain follows the same paradigm as the PI controller in that the derivative term compensates for the velocity of the system. In the Laplace domain, this comes in the form of adding an s term to the derivative gain. In the time domain, this consists of multiplying a gain value by the time derivative of the error as a function of time as seen in Eqn. 9 [2].

$$D_{CL}(s) = K_p + K_d s$$

$$u(t) = K_p e(t) + K_d \dot{e}(t) \tag{9}$$

When modeled in Simulink, this controller looks as shown in Fig. 14 where the inputs are position and velocity of the inverted link. The individual gains are multiplied and then a new voltage is calculated to be fed back to the DC motor.

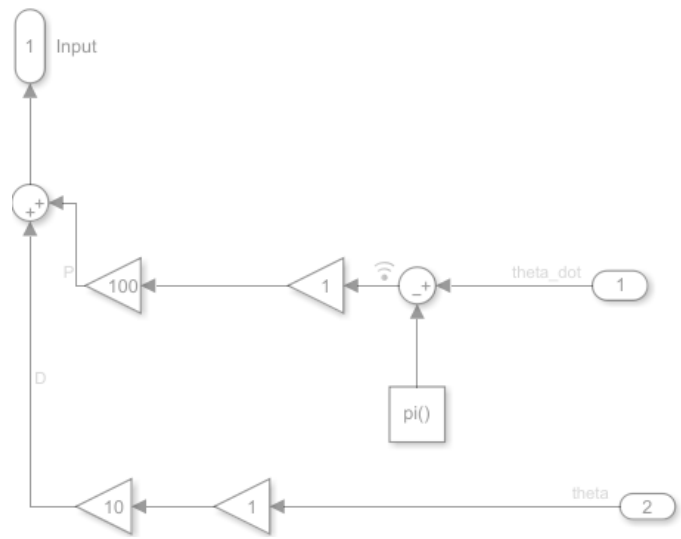

Fig. 14. Proportional and Derivative Controller

4) Proportional, Integral, Derivative (PID)

Continuing with the same paradigm as the previous controllers, the PID controller combines the proportional, integral, and derivative terms in order to tune and account for position, velocity, and acceleration. Although this controller look different mathematically in the Laplace domain versus the time domain, the functionality is the same. Eqn. 10 below shows the mathematical implementation of the PID controller.

$$D_{CL}(s) = K_p + Ki\frac{1}{s}K_d s$$

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, dt + K_d \dot{e}(t) \tag{10}$$

Fig. 15 shows the PID controller as modeled in Simulink and implemented into the feedback loop. This controller accepts inputs, appropriately applies the gain values and then outputs a new voltage value to the DC motor in an attempt to balance the system and achieve steady-state.

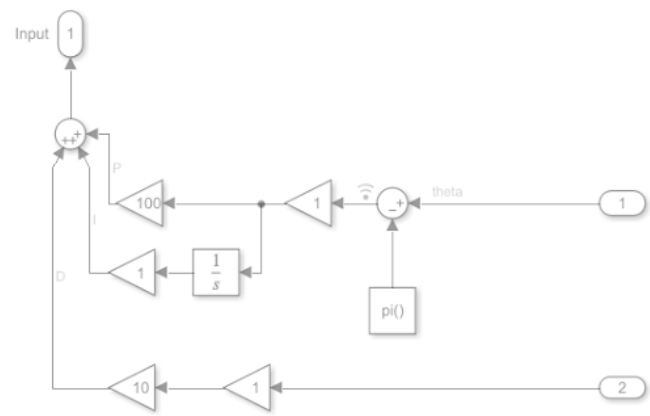

Fig. 15. Proportional, Integral, Derivative Controller

5) PD + Feedforward

As seen in Eqn. 11, the PD plus Feedforward controller takes on a similar form a regular PD controller but incldues other components to account for the feedforward aspect. The M, C, g matrices, which are derived from the dynamical analysis are used in this controller as well to compensate for the torque required to maintain steady-state. The concept of this controller is that the controller will calculate the torque required to achieve the desired position and then adjust the P and D gains accordingly in order to achieve that pose faster and more accurately. However, due to the nature of the inverted pendulum, there is zero torque required to keep the pendulum in the ideal position and therefore this controller realistically acts as a regular PD controller.

$$\tau = Kp\tilde{q} + Kv\tilde{q} - M(q_d)\ddot{q_d} + C(q_d, \dot{q_d})\dot{q_d} + g(q_d) \quad (11)$$

Due to the previously explained nature of this feedforward controller, the Simulink model is identical to that of the PD controller and therefore can be seen in Fig. 9 above.

6) Linear-Quadratic Regulator (LQR)

A linear-quadratic regulator (LQR) controller is an optimized controller where the feedback gains are designed through an optimization process. The performance criterion is minimized using a control law as seen below in Eqn. 12 and Eqn. 13 [2] .

$$u = -k_*^T x \quad (12)$$

$$k_* = \frac{1}{R} b^T P \quad (13)$$

The P term in Eqn. 13 is a matrix in which satisfies the Ricatti equation, a first-order differential equation that is quadratic [2].

$$A^T P + PA - \frac{1}{R} Pbb^T P + Q = 0 \quad (14)$$

When the correct criteria for the above are followed, the performance function, Eqn. 15 [2], will be quadratic and the control law linear, therefore producing the desired controller.

$$J = \int_0^\infty x^T(t)Qx(t) + Ru^2(t)\, dt \quad (15)$$

The LQR controller as modeled in Simulink is shown below in Fig. 16. Similar to the previous controllers, this LQR system is implemented in the place of the controller in the feedback loop.

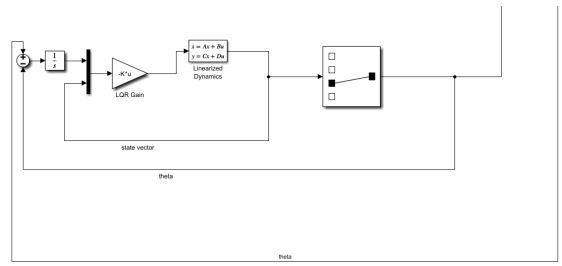

Fig. 16. LQR Controller

7) Model Predictive Control (MPC)

MPC is an advanced method of control, outside of the scope of this project. However, due to previously mentioned reasons, utilizing MPC can produce excellent results, especially in a second order under-actuated system such as the inverted pendulum.

Model predictive control works by using previous inputs and outputs to create predictions of future inputs and outputs. Similar to LQR, control using MPC is optimized by minimizing an objective function. The general form of the objective function is shown in Eqn. 16 [8].

$$\begin{aligned} J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} (\delta(j)[\hat{y}(t+j|t) - w(t+j)]^2] \\ + \sum_{j=1}^{N_u} (\lambda(j)[\Delta u(t+j-1)]^2 \end{aligned} \quad (16)$$

Additionally, MPC will solve an optimization problem at each time step of the simulation in order to find the best action to reach the desired result [7]. The MPC block in Simulink accepts measured outputs and a reference signal as the inputs into the block. This block is also capable of accepting a measured disturbance to take into account. The MPC block integrated into a Simulink model is seen in Fig. 17.

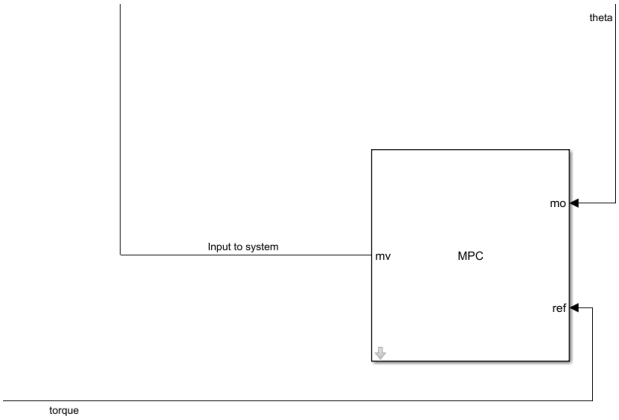

Fig. 17. MPC Controller

Similarly, the MPC block inserted into the feedback loop and plant from the MATLAB provided example is seen below in Fig. 18 [6].

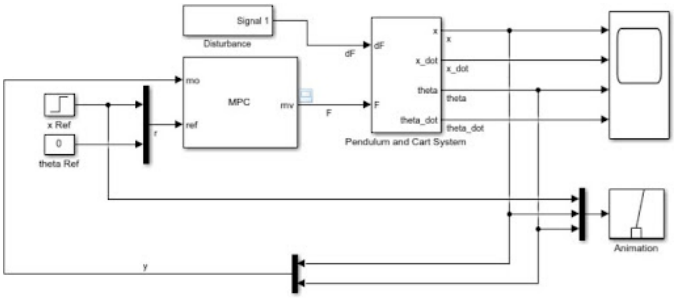

Fig. 18. MPC Controller in MATLAB Provided Example

I. Evaluation of Controllers

Due to the number of controllers being implemented into the system and evaluated, it was deemed imperative to find a consistent and seamless method as evaluating each controller's performance. Using a "To Workspace" block in Simulink, the results of each controller and the associated times are exported to the MATLAB workspace. Here, their data can be broken down by time step of the simulation and different parameters of each controller can be extracted. This process is seen in the MATLAB code snippet below.

```
P = stepinfo(out.simout.Data,out.simout.
    Time,'SettlingTimeThreshold',0.01);
PD = stepinfo(out.simout.Data,out.simout.
    Time,'SettlingTimeThreshold',0.01);
PI = stepinfo(out.simout.Data,out.simout.
    Time,'SettlingTimeThreshold',0.01);
PID = stepinfo(out.simout.Data,out.simout
    .Time,'SettlingTimeThreshold',0.01);
PIDFF = stepinfo(out.simout.Data,out.
    simout.Time,'SettlingTimeThreshold
    ',0.01);

PRiseTime = extractfield(P, 'RiseTime');
PDRiseTime = extractfield(PD, 'RiseTime')
    ;
PIRiseTime = extractfield(PI, 'RiseTime')
    ;
PIDRiseTime = extractfield(PID, 'RiseTime
    ');
PIDFFRiseTime = extractfield(PIDFF, '
    RiseTime');

PSettlingTime = extractfield(P, '
    SettlingTime');
PDSettlingTime = extractfield(PD, '
    SettlingTime');
PISettlingTime = extractfield(PI, '
    SettlingTime');
PIDSettlingTime = extractfield(PID, '
    SettlingTime');
PIDFFSettlingTime = extractfield(PIDFF, '
    SettlingTime');

POvershoot = extractfield(P, 'Overshoot')
    ;
PDOvershoot = extractfield(PD, 'Overshoot
    ');
PIOvershoot = extractfield(PI, 'Overshoot
    ');
PIDOvershoot = extractfield(PID, '
    Overshoot');
PIDFFOvershoot = extractfield(PIDFF, '
    Overshoot');

ThetaRiseTime = [PRiseTime(1); PDRiseTime
    (1); PIRiseTime(1); PIDRiseTime(1);
    PIDFFRiseTime(1)];
ThetaDotRiseTime = [PRiseTime(2);
    PDRiseTime(2); PIRiseTime(2);
    PIDRiseTime(2); PIDFFRiseTime(2)];
ThetaDotDotRiseTime = [PRiseTime(3);
    PDRiseTime(3); PIRiseTime(3);
    PIDRiseTime(3); PIDFFRiseTime(3)];

ThetaSettlingTime = [PSettlingTime(1);
    PDSettlingTime(1); PISettlingTime(1);
    PIDSettlingTime(1); PIDFFSettlingTime
    (1)];
ThetaDotSettlingTime = [PSettlingTime(2);
     PDSettlingTime(2); PISettlingTime(2);
     PIDSettlingTime(2); PIDFFSettlingTime
    (2)];
ThetaDotDotSettlingTime = [PSettlingTime
    (3); PDSettlingTime(3); PISettlingTime
    (3); PIDSettlingTime(3);
    PIDFFSettlingTime(3)];
```

```
ThetaOvershoot = [POvershoot(1);
    PDOvershoot(1); PIOvershoot(1);
    PIDOvershoot(1); PIDFFOvershoot(1)];
ThetaDotOvershoot = [POvershoot(2);
    PDOvershoot(2); PIOvershoot(2);
    PIDOvershoot(2); PIDFFOvershoot(2)];
ThetaDotDotOvershoot = [POvershoot(3);
    PDOvershoot(3); PIOvershoot(3);
    PIDOvershoot(3); PIDFFOvershoot(3)];

controllers = ["P";"PD";"PI";"PID";"PIDFF
    "]
comparison = table(controllers,
    ThetaRiseTime, ThetaSettlingTime,
    ThetaOvershoot, ThetaDotRiseTime,
    ThetaDotSettlingTime,
    ThetaDotOvershoot, ThetaDotDotRiseTime
    , ThetaDotDotSettlingTime,
    ThetaDotDotOvershoot)
```

Using this data extraction and manipulation technique allows for the controller data to be extracted without human error. The rise time, settling time, and overshoot for each controller are evaluated and placed into a table regarding the position (θ), the velocity $\dot{\theta}$, and the acceleration ($\ddot{\theta}$). It should be noted that the MATLAB provided example of the MPC controller does not provide acceleration results and therefore this data is ignored.

This method of evaluation allows the team to automatically extract the rise time, settling time, and overshoot data of each controller and place it into a table for a visualization. These criteria were deemed to be important when evaluating a controller's performance. From a more qualitative perspective, the team also evaluated the controller's by means of how easy they were to implement into the system and how difficult they were to tune. Additionally, the controllers were overall ranked by their ability to reach and maintain steady-state, a balanced inverted pendulum.

IV. Results

After the system was modeled and the controllers implemented, the simulation was run for five seconds. Each controller was tested using a variant subsystem. The following sections show the obtained results of the simulation running. Please see the video presentation for videos of each controller acting upon the system.

A. Controller Responses

The following subsections explore the results of the output link position and associated velocity by controller.

1) P Controller

Fig. 19 shows the obtained results of the position and velocity data when the P controller was implemented into the feedback loop.

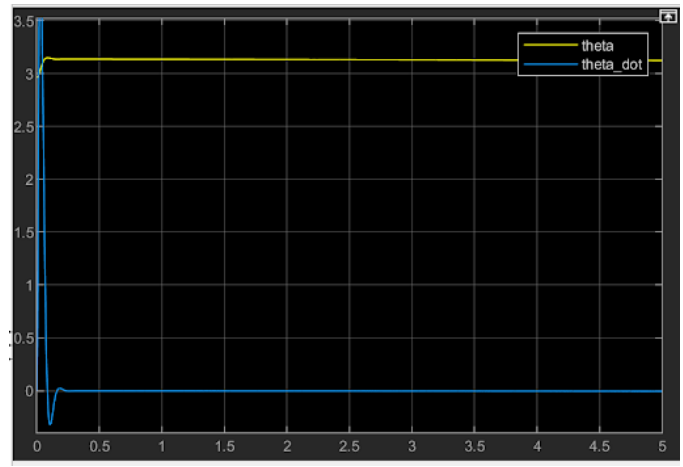

Fig. 19. P Controller Response

As seen, the velocity has a sharp increase in velocity at the beginning and then slight overshoot before reaching steady-state whereas the position smoothly and quickly reaches the target position.

2) PD Controller

Fig. 20 shows the obtained results of the position and velocity data when the P controller was implemented into the feedback loop.

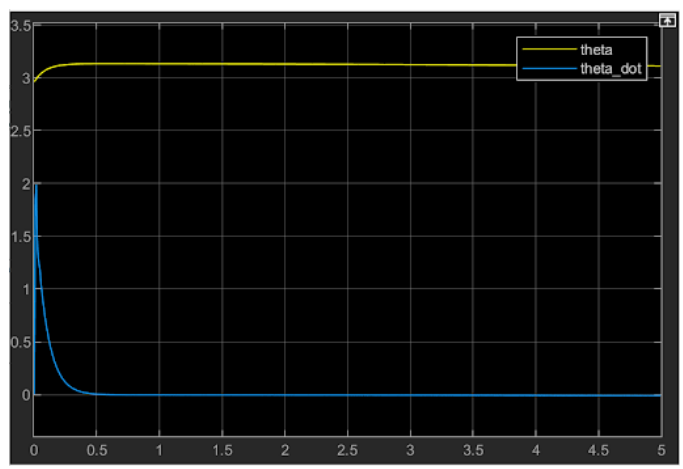

Fig. 20. PD Controller Response

Again, there is a sharp increase in angular velocity at the beginning of the simulation before steady-state is achieved. However, the position again quickly reaches steady state with minimal overshoot.

3) PI Controller

Fig. 21 shows the obtained results of the position and velocity data when the P controller was implemented into the feedback loop.

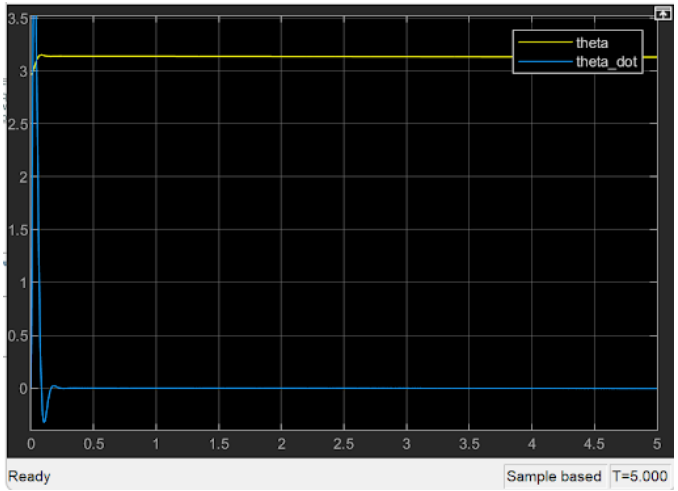

Fig. 21. PI Controller Response

Similar to the P controller, there is again a drastic increase in velocity and then negative overshoot before the steady-state performance is reached.

4) PID Controller

Fig. 22 shows the obtained results of the position and velocity data when the P controller was implemented into the feedback loop.

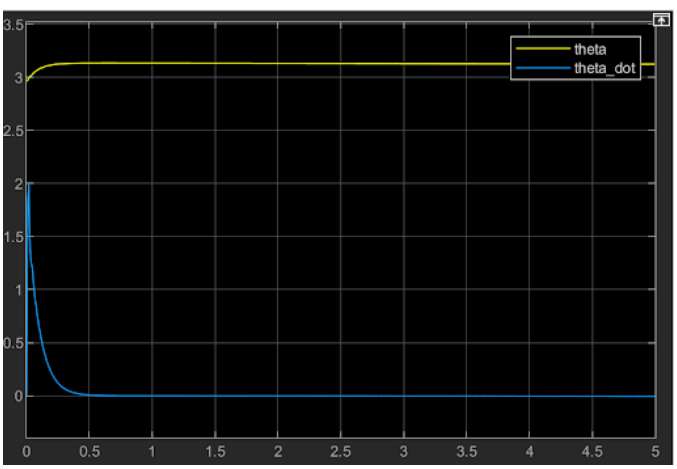

Fig. 22. PID Controller Response

Likewise to the PD controller, the velocity has a sharp increase at the beginning but quickly recovers and reaches the desired velocity while the position pose is reached with minimal rise time and overshoot.

5) PD with Feedforward Controller

Fig. 23 shows the obtained results of the position and velocity data when the P controller was implemented into the feedback loop.

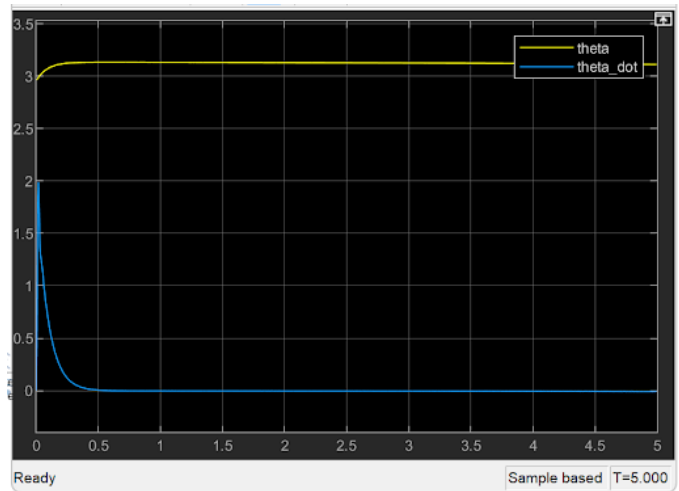

Fig. 23. PD + Feedforward Controller Response

Again, the velocity shows a sharp increase at the beginning of the simulation but then recovers quickly and before reaching the steady-state that is desired.

6) LQR Controller Results

Although under-developed, the LQR controller was implementing into the feedback loop acting on the plant. Figs. 24, 25, and 26 show the position, velocity, and acceleration responses due to the LQR controller.

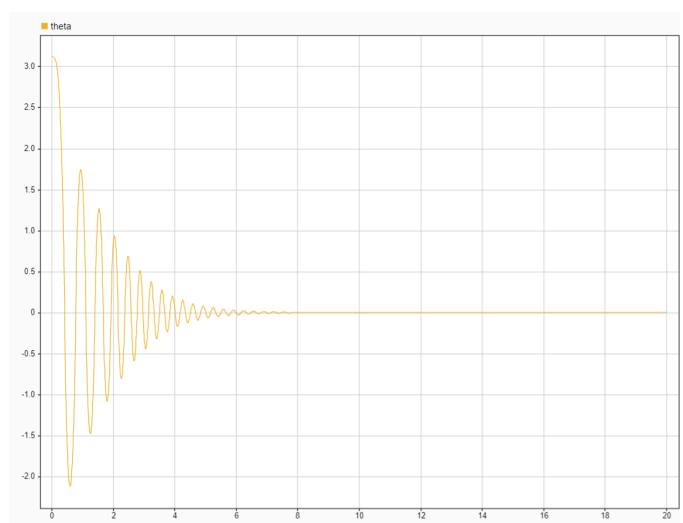

Fig. 24. LQR Controller Response - θ

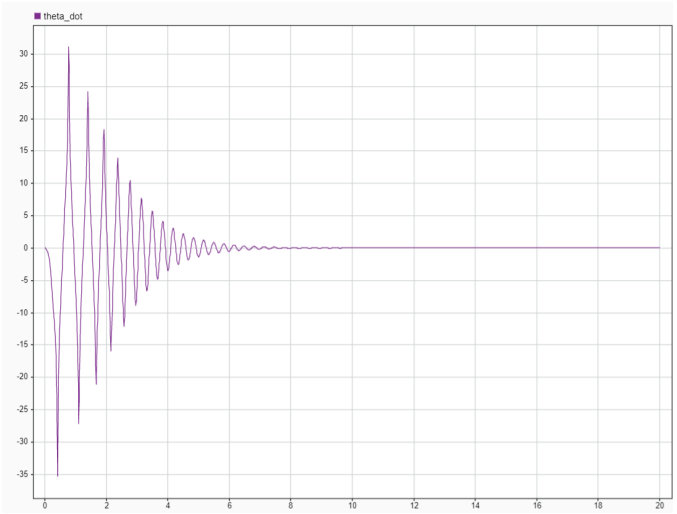

Fig. 25. LQR Controller Response - $\dot{\theta}$

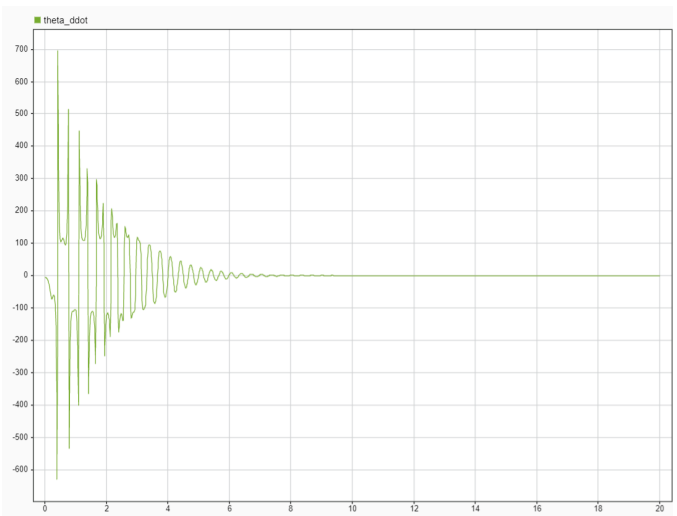

Fig. 26. LQR Controller Response - $\ddot{\theta}$

Although this controller is also successful in achieving steady-state performance, there is a lot of oscillation at the beginning before slowly settling to stability.

B. MPC Controller Results

By running the MATLAB provided example of the inverted pendulum with an MPC controller, the position and velocity response plots are generated. Fig. 27 below shows the linear and angular position and velocity responses to this controller. Note the second 'bump' roughly halfway through the simulation due to an external disturbance being applied.

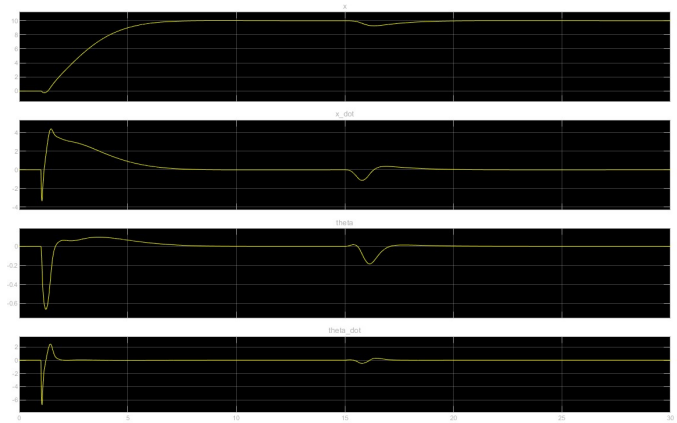

Fig. 27. MPC Controller Results

C. Controller Evaluation

After the controllers are implemented, they must also be tuned in order to achieve optimal results in balancing the system. The following table, Tbl. I, shows the gain values used for each error based controller. These values were used in the controllers at the time of the simulation and produced the results seen above.

TABLE I
K_p, K_i, AND K_d BY CONTROLLER

Controller	K_p	K_i	K_d
P	15	-	-
PD	15	-	1
PI	15	1	-
PID	15	1	1
PD+FF	15	-	1

Tbl. II below explores the data of rise time, settling time, and overshoot percentage regarding the angular position of the inverted link by controller type.

TABLE II
RISE TIME, SETTLING TIME, AND OVERSHOOT OF θ BY CONTROLLER

Controller	θ Rise Time	θ Settling Time	θ Overshoot
P	0.0	0.18681	2.4638
PD	0.0	0.10707	0.72551
PI	0.0	0.18169	2.2631
PID	0.0	0.13199	0.13199
PID+FF	0.0	0.13199	0.31653
LQR	0.390	4.94	36.69
MPC	0.2436	4.9793	47.27

Tbl. III shows the data of rise time, settling time, and overshoot percentage regarding the angular velocity of the inverted link per controller.

TABLE III
RISE TIME, SETTLING TIME, AND OVERSHOOT OF $\dot{\theta}$ BY CONTROLLER

Controller	$\dot{\theta}$ Rise Time	$\dot{\theta}$ Settling Time	$\dot{\theta}$ Overshoot
P	0.13245	4.9984	17359
PD	3.0474	4.9911	6.1243
PI	0.13261	4.992	22421
PID	0.2.6587	4.9962	15.613
PID+FF	0.2.6587	4.9962	15.613
LQR	0.17	5.51	34.6
MPC	0.0001324	4.7403	2127.7

Tbl. IV provides the data of rise time, settling time, and overshoot percentage regarding the angular velocity of the inverted link broken down by controller type.

TABLE IV
RISE TIME, SETTLING TIME, AND OVERSHOOT OF $\ddot{\theta}$ BY CONTROLLER

Controller	$\ddot{\theta}$ Rise Time	$\ddot{\theta}$ Settling Time	$\ddot{\theta}$ Overshoot
P	0.00000351	5.0	356430
PD	0.0000146	5.0	70668
PI	0.00000167	5.0	747610
PID	0.00000969	4.99	107220
PID+FF	0.00000969	4.99	107220
LQR	0.185	5.742	618.38
MPC	N/A	N/A	N/A

V. DISCUSSION

All of the controllers presented were capable of maintaining the stability of the inverted pendulum system with varying levels of performance. The basic error-based controllers performed especially well, with minimal rise time, settling time, and overshoot for the pendulum position. Although we implemented both PD and PD+Feedforward controllers, these are functionally identical in the case where desired acceleration and velocity are zero.

A. Significance of Results

Using the plots in the previous section, conclusions about each controller can be drawn. As noted, every controller that was implemented is capable of controlling the system to the point of steady-state and stability within the five second simulation window. In all cases of the error based controllers, the position is quickly and smoothly achieved due to the sharp increase in velocity. This link overcomes to 10 degrees offset quickly and is successful in maintaining this position for the result of the simulation. The sharp increases in velocity at the beginning of each controller represents the system trying to rapidly overcome the angle offset by applying a high voltage immediately. This in turn causes the link to rapidly move in the opposite direction of the motion of the base and therefore achieve the desired pose. Once this pose is reached, the velocity needs to quickly decrease to compensate for movement in the opposite direction without over stimulating the system and causing the link to move into the negatives. As seen from the controller performance controllers above, the acceleration and velocity profiles tend to have a much larger overshoot. This is due to the motor quickly being input with more voltage in order to drive the cart. The system's inertia must be overcome to allow for movement and the cart must move quickly in order to balance the inverted link.

The LQR controller was also successful in balancing the system and allowing the inverted pendulum to balance. The high level of oscillation at the beginning of the simulation is due to the quadratic nature of the controller. Once this controller becomes linear, the steady-state is achieved. With further exploration and research, this controller can be tuned better to remove this oscillation.

Due to the MPC controller being implemented from MATLAB, the results are excellent. The position and velocity smoothly and quickly approach the steady-state values without too much overshoot. The additional bumps later in the simulation are due to an external disturbance and it is seen that the system quickly recovers back to the stable state. Again, with further research, this controller can also be expanded upon and implemented better into our modeled system.

Based on the collected data and the performance of each controller, they are ranked in terms of performance taking all parameters into account. The following list shows the ranked controllers from best to worst.

1) Model Predictive Control (MPC)
2) Proportional and Derivative (PD) / PD plus Feedforward
3) Proportional, Integral, Derivative (PID)
4) Proportional (P) / Proportional and Integral (PI)
5) Linear-Quadratic Regulation (LQR) (See future work section)

B. Challenges

This problem presented several challenges worth presenting. The first was the challenge introduced by tuning the controller gains. While error-based controllers are relatively trivial to implement, tuning multiple gain values can be a challenge. The controllers presented were tuned manually, and the performance presented is not guaranteed to be ideal. The advanced controllers have an advantage over the basic ones in this area.

The dynamics of the inverted pendulum also introduced challenges. The inverted pendulum is an under-actuated system, with the horizontal force on the cart controlling both cart motion and pendulum motion. The error-based controllers are only designed to control a single output, which in this case was pendulum position. A simulation using any of the error-based controllers shows a constant cart motion in the direction of the original offset. To remedy this, the cart position would have to be incorporated into the controller.

The challenges presented by the dynamical model scale exponentially when applied to a double inverted pendulum. While controlling a double inverted pendulum was an initial goal for this paper, that was quickly abandoned when we

realized the complexity introduced by another under-actuated link as well as another output variable to be controlled.

The final challenge was technological. The inverted pendulum was simulated using a combination of the simulink, simscape multibody, and simscape electrical frameworks. These tools were unfamiliar to several members of our group, and presented a steep learning curve.

C. Advanced Controllers

While the PID controller was adequate for the problem of inverting the double pendulum, other methods were also considered for implementation. These methods however were beyond the scope of the class or could not be learned within the time frames for this project and the course, however they will be discussed in this section

Other options for controlling the double inverted pendulum include linear quadratic regulator (LQR), model predictive control (MPC), fuzzy control, root locus, computed torque, PD plus feed forward, impedance control and compliance control, and others. Since the other methods of control are numerous this paper will only discuss LQR and MPC, two popular alternatives to PID control.

Another type of advanced controlling option is the model predictive controller (MPC). As described by Camacho, model predictive control is not a specific strategy but consists of multiple methods to design the controller by minimizing an objective function.

An oversimplified analogy of MPC control from [8] is illustrated in Fig. 28 by a driver in a moving car. The driver has to look ahead to a short horizon in order to predict what actions will take place next. This is similar to model predictive control. If the driver were only driving using the rear windows, this is similar to PID control which changes input based on errors. The author admits this difference is easily overcome if future set points are used as the inputs in the PID controller. The analogy is illustrated in the figure below.

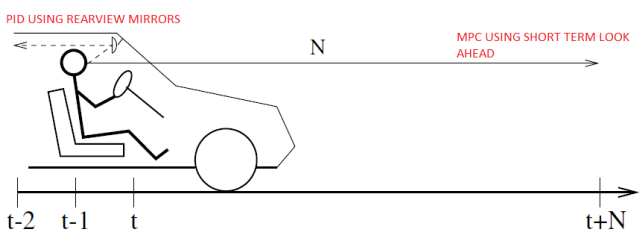

Fig. 28. MPC Controller vs PID Controller

References

[1] A. Benitez, "Inverted Pendulum," Free CAD Designs, Files amp; 3D Models — The GrabCAD Community Library, https://grabcad.com/library/inverted-pendulum-4 (accessed Jul. 24, 2023).

[2] Spong, Mark W., et al. *Robot Modeling and Control.* Wiley, 2020.

[3] M. Jibril, M. Tadese, and F. Bogale, "Stability Analysis of an Inverted Pendulum on a Cart with the Presence of Restoring and Frictional Forces Disturbance using Observer Based and Full State Feedback H2 Controllers," philarchive.org, https://philarchive.org/archive/JIBSAO (accessed Jul. 27, 2023).

[4] MIT, "Underactuated Robotics," Ch. 8 - Linear Quadratic Regulators, http://underactuated.mit.edu/lqr.html (accessed Jul. 26, 2023).

[5] Brigham Young University, "The Inverted Pendulum."

[6] MathWorks, "Control of an Inverted Pendulum on a Cart," MATLAB amp; Simulink, https://www.mathworks.com/help/mpc/ug/control-of-an-inverted-pendulum-on-a-cart.html(accessed Jul. 24, 2023).

[7] MathWorks, "What is Model Predictive Control?," What is Model predictive control? - matlab amp; simulink, https://www.mathworks.com/help/mpc/gs/what-is-mpc.html (accessed Jul. 24, 2023).

[8] Camacho, Eduardo F., and Carlos. Bordons Alba. *Model Predictive Control*. Springer London, 2007.

VI. Author Biographies

Christian Chang is a student in the Masters program for Robotics Engineering at Worcester Polytechnic Institute and has his bachelor degree from University of Southern California. His professional background is in transportation/infrastructure engineering and he hopes to apply his knowledge in Robotics to future transportation needs.

Aaron Erickson is a Software Engineer working on real time operating systems. He also has experience as a conceptual, structures, subsystems, and mechanisms engineer. He received his undergrad degree in mechanical engineering with a concentration in Mechatronics from Cal Poly Slo.

Joshua Gross is a software engineer at Modern Technology Solutions Inc. (MTSI) where he focuses on the creation of leading edge software for the United States Air Force. He graduated from West Virginia University with a Bachelor of Science in Mechanical Engineering. Joshua is currently pursuing his Master of Science in robotics engineering from Worcester Polytechnic Institute with hopes of focusing on pediatric prosthetics.

Will Yingling is employed as a software engineer with Spectra Logic Inc. He graduated from Colorado State University in 2019 with Bachelor of Science in Computer Science and a minor in mathematics. He is currently pursuing his Masters of Science in Robotics Engineering at Worcester Polytechnic Institute.